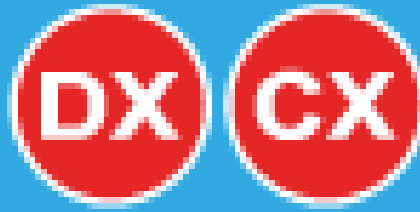


CodeRage® XI

Productivity, Platforms, Performance

15-17 November 2016



FASTER PARALLEL PROGRAMS WITH IMPROVED FASTMM



Primož Gabrijelčič
@thedelphigeek

<http://primoz.gabrijelcic.org>



INFO

Slides and code are available at
<http://thedelphigeek.com/p/presentations.html>

HISTORY

- Developed by Pierre LeRiche for the FastCode project
 - <https://en.wikipedia.org/wiki/FastCode>
 - Version 4, hence FastMM4
- Included in RAD Studio since version 2006
 - <http://www.tindex.net/Language/FastMMmemorymanager.html>
- Much improved since
 - Don't use default FastMM, download the fresh one
 - <https://github.com/pleriche/FastMM4>

FEATURES

- Fast
- Fragmentation resistant
- Access to > 2GB
- Simple memory sharing
- **Memory leak reporting**
- **Catches some memory-related bugs**

PROBLEMS

- Can be slow in a multithreaded environment

INTERNALS

TOP VIEW

- Three memory managers in one
- **Small blocks (< 2,5 KB)**
 - Most frequently used (99%)
 - Medium blocks, subdivided into small blocks
- **Medium blocks (2,5 – 260 KB)**
 - Allocated in chunks (1,25 MB) and subdivided into lists
- **Large blocks (> 260 KB)**
 - Allocated directly by the OS

DETAILS

- One large block allocator
- One medium block allocator
- Multiple (54+2) small block allocators
 - SmallBlockTypes
 - Custom, optimized Move routines (FastCode)
- Each allocator has its own lock
 - If SmallAllocator is locked, SmallAllocator+1 or SmallAllocator+2 is used

REASONS FOR SLOWDOWN

- Threads are fighting for allocators
- Solution – Change the program
 - Hard to find out the problematic code

DEMO

- Steve Maughan
<http://www.stevemaughan.com/delphi/delphi-parallel-programming-library-memory-managers/>
- Redesigned to use OmniThreadLibrary

DIAGNOSING FASTMM BOTTLENECKS

FASTMM4 LOCKING

```
if IsMultiThread then begin
    while LockCmpxchg(0, 1, @MediumBlocksLocked) <> 0 do begin
        {$ifdef NeverSleepOnThreadContention}
        {$ifdef UseSwitchToThread}
            SwitchToThread; //any thread on the same processor
        {$endif}
        {$else}
            Sleep(InitialSleepTime); //0; any thread that is ready to run
            if LockCmpxchg(0, 1, @MediumBlocksLocked) = 0 then
                Break;
            Sleep(AdditionalSleepTime); //1; wait
        {$endif}
    end;
end;
```

LOCK CONTENTION LOGGING

```
LockMediumBlocks({$ifdef LogLockContention}LDidSleep{$endif});
```

```
ACollector := nil;
```

```
{$ifdef LogLockContention}
```

```
if LDidSleep then
```

```
    ACollector := @MediumBlockCollector;
```

```
{$endif}
```

```
if Assigned(ACollector) then begin
```

```
    GetStackTrace(@LStackTrace, StackTraceDepth, 1);
```

```
    ACollector.Add(@LStackTrace[0], StackTraceDepth);
```

```
end;
```

FASTMM4DATACOLLECTOR

- Opaque data
- Completely static
 - Can't use MM inside MM
 - Agreed max data size
- Most Frequently Used
- Generational
 - Reduce the problem of local maxima
 - Two generations, sorted
 - Easy to expand to more generations

OUTPUT

- Results for all allocators are merged
- Top 10 call stacks are written to
`<programname>_MemoryManager_EventLog.txt`

FINDINGS

IT IS HARD TO RELEASE MEMORY

- GetMem does not represent a problem
 - It can (with small blocks) upgrade to unused allocator
 - One thread doesn't block another
- Time is mostly wasted in FreeMem
 - FreeMem **must** use allocator that produced the memory
 - One thread blocks another

SOLUTION

PARTIAL SOLUTION

- If allocator is locked, delay the FreeMem
- Memory block is pushed on a 'to be released' stack
- Each allocator gets its own "release stack"

```
while LockCmpxchg(0, 1, @LPSThresholdType.BlockTypeLocked) <> 0 do begin
  {$ifdef UseReleaseStack}
    LPReleaseStack := @LPSThresholdType.ReleaseStack;
    if (not LPReleaseStack^.IsFull) and LPReleaseStack^.Push(APointer) then
      begin
        Result := 0;
        Exit;
      end;
  {$endif}
end;
```

- When allocator is successfully locked, all memory from its release stack is released.

FASTMM4LOCKFREESTACK

- Very fast lock-free stack implementation
 - Taken from OmniThreadLibrary
- Windows only
- Dynamic memory
 - Allocated at startup
 - Uses HeapAlloc for memory allocation

PROBLEMS

- Release stacks work, but not perfectly
1. FreeMem can still block if multiple threads are releasing similarly sized memory blocks.
 - Solution: Hash all threads into a pool of release stacks.
 2. Somebody has to clean after terminated threads.
 - Solution: Low-priority memory release thread.
 - Currently only for medium/large blocks.
 - CreateCleanupThread/DestroyCleanupThread

FULL SOLUTION?

```
while LockCmpxchg(0, 1, @LPSThreadType.ThreadTypeLocked) <> 0 do
begin
{$ifdef UseReleaseStack}
    LPReleaseStack := @LPSThreadType.ReleaseStack[GetStackSlot];
    if (not LPReleaseStack^.IsFull) and LPReleaseStack^.Push(APointer)
    then begin
        Result := 0;
        Exit;
    end;
{$endif}
```

- GetStackSlot hashes thread ID into [0..NumStacksPerBlock-1] range

BUNCH OF RELEASE STACKS!

- 56 + 1 + 1 allocators, each with 64 release stacks
 - Each release stack is very small
 - 36 static bytes
 - 88 dynamic bytes (16 pointers per stack)
- In 32-bit world
 - $58 * 64 * (36 + 88) = 460 \text{ KB}$

IMPROVE YOUR CODE

DEPLOYMENT

- Main FastMM repository
 - <https://github.com/pleriche/FastMM4>
- Define LogLockContention
or
- Define UseReleaseStack
- Rebuild

Q & A

Slides and code are available at
<http://thedelphigeek.com/p/presentations.html>